

Performance Tools and Optimization for Blue Gene/L

Example : GTC (Princeton Plasma Physics Lab)

Bob Walkup (walkup@us.ibm.com)
IBM Thomas J. Watson Research Center

With special thanks to Stephane Ethier
Princeton Plasma Physics Laboratory

1. Check computation vs. communication time

Tools: MPI wrappers such as libmpitrace_c/f.a
or libmpihpm_c/f.a (adds perf counters)

2. If compute bound, get more detailed information

Tools: Xprofiler, gprof, libmpitrace_c/f.a

3. Check code generation (-qlist -qsource), and tune the code

Tools: the compiler, libraries (MASS, MASSV)

Communication vs. Computation

Link with libmpitrace_c/f.a; run the code.

or link with libmpihpm_c/f.a -lbg1_perfctr.rts; run the code
(starts counters in mpi_init, stops them in mpi_finalize)

```
elapsed time from clock-cycles using freq = 700.0 MHz
-----
MPI Routine          #calls    avg. bytes   time(sec)
-----
MPI_Comm_size         3          0.0        0.000
MPI_Comm_rank         3          0.0        0.000
MPI_Recv              24         24.0       0.000
MPI_Sendrecv         2800      208777.8     4.438
MPI_Bcast             3          85.3       0.000
MPI_Gather            1626       104.2      0.253
MPI_Reduce            36         207.2      0.002
MPI_Allreduce         675       77040.1     8.507
-----
total communication time = 13.201 seconds.
total elapsed time      = 384.498 seconds.
top of the heap address = 81.703 MBytes.
```

...

```
BG/L counters avg Flops = 146.738 MFlops
```

Communication summary for all tasks:

```
minimum communication time = 6.665 sec for task 222
median   communication time = 9.656 sec for task 371
maximum  communication time = 14.052 sec for task 40
```

For this application, the focus is on computation.

BGL Counter Statistics

Link with libmpihpm_c/f.a -lbgl_perfctr.rts

Counter	Avg_Value	Min_Value	Min_Rank	Max_Vaule	Max_Rank
CYCLES	2.6871e+11	2.6870e+11	207	2.6915e+11	0
ADD+SUB	8.7109e+09	8.6736e+09	500	8.7519e+09	176
MUL+DIV	1.3175e+10	1.3153e+10	389	1.3201e+10	225
FMADD	1.7267e+10	1.7241e+10	414	1.7293e+10	170
FPMADD	1.0704e+05	1.0704e+05	3	1.0704e+05	3
LD_DOUBLE	7.4458e+09	7.3934e+09	500	7.5010e+09	176
ST_DOUBLE	3.8258e+09	3.8110e+09	389	3.8417e+09	225
LD_QUAD	9.6443e+09	9.6221e+09	222	9.6661e+09	74
ST_QUAD	9.7205e+09	9.7094e+09	499	9.7314e+09	47
L3_HIT	2.4224e+09	2.2870e+09	240	2.5209e+09	342
L3_MISS	3.8211e+08	3.8035e+08	419	3.8307e+08	176
XM_PKTS	1.9073e+05	8.5200e+02	0	2.1803e+05	1
XP_PKTS	1.9069e+05	8.8000e+02	7	2.1797e+05	4
YM_PKTS	1.1751e+06	1.1406e+06	391	1.2342e+06	0
YP_PKTS	1.5518e+06	1.5193e+06	120	1.6583e+06	504
ZM_PKTS	1.5031e+05	3.0520e+03	8	8.3984e+05	322
ZP_PKTS	1.9601e+05	1.9550e+03	480	1.2993e+06	378

Average MFlops = 146.980 per MPI task

Total GFlops = 75.254

BGL FP op count = (add+sub) + (mul+div) + 2*fmadd + 4*fpmadd

MFlops/GFlops values are suspect due to counter limitations.

FPU and LSU averages are per MPI process.

L3 and packet averages are per BGL node.

Blue Gene has many more counters, but interpretation is not simple. You can use other tools, such as Jim Sexton's APC, to get more counter data.

Overall “report card” – from Power5 Counters

CPI = 1.29 clock cycles per instruction completed
IPC = 0.78 instructions completed per clock cycle

Instruction mix:

29.16 % loads (load references)/(instructions completed)
19.15 % floating-point loads (fp loads)/(instructions completed)
9.64 % stores (store references)/(instructions completed)
5.42 % floating-point stores (fp stores)/(instructions completed)
33.92 % floating-point operations (fp ops)/(instructions completed)
36.57 % integer operations (fx ops)/(instructions completed)
3.71 % branches (branches issued)/(instructions completed)

Cache and TLB:

3.15 % loads that miss L1 Data cache / (load refs)
19.60 % stores that miss L1 Data cache / (store refs)
0.03 % loads that miss TLB / (load refs)
1.74 % loads from L2 / (load refs)
0.06 % loads from L3 / (load refs)
0.13 % loads from memory / (load refs)
2.51 % cycles in data "table-walks" / (run_cycles)

Floating Point:

700.4 MFlops per process
10.57 % theoretical floating-point peak
13.84 % hardware floating-point peak (fp_ops / 2*run_cycles)
38.95 % fp ops that are fmadds (fmadds/fp_ops)
0.78 % fp ops that are divides (fp divides/fp_ops)
6.41 % max fraction of cycles in division
0.33 % fp ops that are square_roots (square-roots/fp_ops)
2.74 % max fraction of cycles in square_roots

Prefetches:

0.59 % L1 prefetches / (load refs)
0.37 % L2 prefetches / (load refs)

Branches:

5.39 % branches mispredicted (mispredictions/branches issued)

Stalls:

25.48 % cycles stalled for loads/stores
14.10 % cycles stalled for L1 data cache miss
39.44 % cycles stalled in the floating-point pipeline
1.59 % cycles stalled for floating-point division or square-roots
4.41 % cycles stalled in the integer unit

Profiling : gprof and Xprofiler

Compile and link with -g -pg

Optionally link with libmpitrace_c/f.a, to limit profiler output.
Get gmon.out for rank 0, and ranks with minimum, median, and maximum communication times. Would get one gmon.out for each app unless you use the MPI wrappers or directly call mondisable().

gprof: gprof your.exe gmon.out.n >gprof_n.txt
Produces a “flat profile” ... subroutine-level, and a call-graph section.

Xprofiler can provide statement-level timing data: clock ticks tied to source lines.

Set your DISPLAY variable (requires X-windows)
Xprofiler your.exe gmon.out.n
Select “Report” then “Flat Profile”
Click on a routine and select “Code Display”, then
“Show Source Code”

Gprof Example: GTC Flat profile

Each sample counts as 0.01 seconds.

%	cumulative	self	self	total	
time	seconds	seconds	calls	s/call	s/call name
37.43	144.07	144.07	201	0.72	0.72 chargei
25.44	241.97	97.90	200	0.49	0.49 pushi
6.12	265.53	23.56			_xldintv
4.85	284.19	18.66			cos
4.49	301.47	17.28			sinl
4.19	317.61	16.14	200	0.08	poisson
3.79	332.18	14.57			_pxldmod
3.55	345.86	13.68			__ieee754_exp
2.76	356.48	10.62			BGLML_Messager_advance
2.23	365.05	8.57	200	0.04	shifti
1.23	369.80	4.75	400	0.01	smooth
0.79	372.85	3.05			exp
0.53	374.88	2.03	200	0.01	field
0.27	375.91	1.03			finitel
0.24	376.84	0.93			TreeAllreduce_fifo
0.24	377.77	0.93			_exp
0.20	378.54	0.77			_sin
0.15	379.12	0.58			readfun_torus
0.12	379.59	0.47	1	0.47	poisson_initial

...

Performance issues are mainly in two routines: chargei and pushi. There are lots of intrinsic functions, and expensive conversions to get the integer part of a floating-point number.

Flat Profile using MPI wrapper libraries

Compile with -qdebug=function_trace

Link with libmpitrace_c/f.a

Run with env variable FLAT_PROFILE=yes

```
Elapsed-time flat profile for MPI task 0.
```

%Time	Self(seconds)	Inclusive	#Calls	Routine
45.05	173.2084	173.2084	201	chargei
43.92	168.8568	168.8568	200	pushi
4.24	16.2962	16.8397	200	poisson
3.64	13.9843	13.9843	200	shifti
1.55	5.9536	5.9883	400	smooth
0.75	2.8856	2.8856	200	field
0.44	1.6768	1.7237	1	load
0.20	0.7638	0.8712	1	snapshot
0.14	0.5435	0.5435	1	poisson_initial
0.03	0.1301	384.6211	1	gtc
0.01	0.0548	0.0852	1	setup
0.01	0.0462	0.0462	1	set_random_zion
0.01	0.0327	0.0428	32155	boozer2x
0.01	0.0321	0.0433	32155	boozer2z
0.01	0.0308	0.0308	2375	spcpft
0.01	0.0268	0.0268	1	read_input_params

Uses compiler-generated hooks for entry/exit of each routine. Get elapsed-time data for routines compiled with the function trace option. No interrupts.

This method can be useful because you get inclusive elapsed times, but Xprofiler is recommended because it provides a lot more information.

Xprofiler Example : GTC pushi routine

```
Line   ticks source
115      do m=1,mi
116    657   r=sqrt(2.0*zion(1,m))
117    136   rinv=1.0/r
118    34    ii=max(0,min(mpsi-1,int((r-a0)*delr)))
119    55    ip=max(1,min(mflux,1+int((r-a0)*d_inv)))
120   194   wp0=real(ii+1)-(r-a0)*delr
121    52    wp1=1.0-wp0
122   104   tem=wp0*temp(ii)+wp1*temp(ii+1)
123    86    q=q0+q1*r*ainv+q2*r*r*ainv*ainv
124   166   qinv=1.0/q
125    68    cost=cos(zion(2,m))
126    18    sint=sin(zion(2,m))
129   104   b=1.0/(1.0+r*cost)
130      g=1.0
131      gp=0.0
134      ri=0.0
135      rip=0.0
136   97    dbdp=-b*b*cost*rinv
137   18    dbdt=b*b*r*sint
      ...

```

clock ticks are 100 per second

Lots of expensive operations : sqrt, division, cos, sin, ...

Blue Gene default routines for sqrt, cos, sin, exp, ... are very slow, from GNU libm.a. Use libmass.a, but better to vectorize and use libmassv.a routines.

For scalar MASS:

link with libmass.a -Wl,--allow-multiple-definition

For vector MASSV, can try -qhot or hand-code.

Adding Vector MASS Routines

```
#ifdef USE_VECTOR
do m = 1, mi
    r_vec(m) = 2.0*zion(1,m)
    z_vec(m) = zion(2,m)
end do
call vssqrt(r_vec, r_vec, mi)
call vsrec(rinv_vec, r_vec, mi)
call vssincos(sin_vec, cos_vec, z_vec, mi)
do m = 1, mi
    rfac = rw*(r_vec(m) - rc)
    z_vec(m) = - rfac**6
end do
call vsexp(z_vec, z_vec, mi)
#endif

do m=1,mi
#ifdef USE_VECTOR
    r = r_vec(m)
    rinv = rinv_vec(m)
#else
    r=sqrt(2.0*zion(1,m))
    rinv=1.0/r
#endif
    ii=max(0,min(mpsi-1,int((r-a0)*delr)))
    ip=max(1,min(mflux,1+int((r-a0)*d_inv)))
    wp0=real(ii+1)-(r-a0)*delr
    wp1=1.0-wp0
    tem=wp0*temp(ii)+wp1*temp(ii+1)
    q=q0+q1*r*ainv+q2*r*r*ainv*ainv
    qinv=1.0/q
#endif
    ifdef USE_VECTOR
        cost = cos_vec(m)
        sint = sin_vec(m)
    else
        cost=cos(zion(2,m))
        sint=sin(zion(2,m))
    endif
    ...

```

Xprofiler Example : GTC chargei routine

```
46      do larmor=1,4
47 1047      rdum=delr*max(0.0,min(a1-a0,r+rhoi*p gyro(larmor,ipjt)-a0))
48 386      ii=max(0,min(mpsi-1,int(rdum)))
49 502      wpl=rdum-real(ii)
50 439      wpion(larmor,m)=wpl
51
52          ! particle position in theta
53 226      tflr=thetatmp+rhoi*t gyro(larmor,ipjt)
54
55          ! inner flux surface
56      im=ii
57 452      tdum=pi2_inv*(tflr-zetatmp*qtinv(im))+10.0
58 327      tdum=(tdum-aint(tdum))*delt(im)
59 526      j00=max(0,min(mtheta(im)-1,int(tdum)))
60 551      jt0n0(larmor,m)=igrid(im)+j00
61 1136      wt0n0(larmor,m)=tdum-real(j00)
62
63          ! outer flux surface
64      im=ii+1
65 164      tdum=pi2_inv*(tflr-zetatmp*qtinv(im))+10.0
66 380      tdum=(tdum-aint(tdum))*delt(im)
67 532      j01=max(0,min(mtheta(im)-1,int(tdum)))
68 395      jt0n1(larmor,m)=igrid(im)+j01
69 801      wt0n1(larmor,m)=tdum-real(j01)
70      enddo
```

Issues : pipelining, type-conversion, register spills, ...

Tuning : check the compiler listing!

Compiler Listing Example : chargei routine

```
Source section
55 | ! inner flux surface
56 | im=ii
57 | tdum=pi2_inv*(tflr-zetatmp*qtinv(im))+10.0
58 | tdum=(tdum-aint(tdum))*delt(im)
59 | j00=max(0,min(mtheta(im)-1,int(tdum)))
60 | jt0n0(larmor,m)=igrid(im)+j00
61 | wt0n0(larmor,m)=tdum-real(j00)

Register section
GPR's set/used:    ssss ssss ssss s-ss    ssss ssss ssss ssss
FPR's set/used:    ssss ssss ssss ssss    ssss ssss ssss ssss
                  ssss ssss ssss ss--    ---- ---- ---s s--s
CCR's set/used:    ssss ssss

Assembler section
58| 000DDC fmr      3      LRFL      fp1=fp15
69| 000DE0 frsp     1      CVLS      fp3=fp3,fcr
69| 000DE4 fmsubs   4      FMSS      fp0=fp3,fp2,fp0,fcr
69| 000DE8 stfsux   0      STFSU     gr3,wtonl(gr3,gr20,0)=fp0
69| 000DEC stw      0      ST4A      #SPILL29(gr31,428)=gr3
49| 000DF0 lfd      0      LFL       fp2=#MX_CONVS1_4(gr31,216)
49| 000DF4 fadd     1      AFL       fp0=fp2,fp17,fcr
49| 000DF8 frsp     4      CVLS      fp0=fp0,fcr
49| 000DFC fmsubs   4      FMSS      fp0=fp0,fp30,fp4,fcr
50| 000E00 stfsux   0      STFSU     gr4,wpon(gr4,gr5,0)=fp0
50| 000E04 stw      0      ST4A      #SPILL52(gr31,520)=gr4
58| 000E08 bl       0      CALLN     fp1=_xldintv,0,fp1,gr1,gr31,...
59| 000E0C mullw    2      M         gr3=gr19,gr15
58| 000E10 rlwinm   1      SLL4     gr5=gr15,2
60| 000E14 mullw    2      M         gr4=gr18,gr15
64| 000E18 addi     1      AI        gr15=gr15,1,ca"
60| 000E1C lwzx     1      L4A      gr0=igrid(gr24,gr4,0)
...
...
```

Issues : pipelining, function call for aint(x), register spills

Tuning : replace aint(x) with real(int(x)), eliminates function call, improves pipelining

GTC Performance on Blue Gene/L

Original code: main loop time = 384 sec (512 nodes, coprocessor)

Tuned code : main loop time = 244 sec (512 nodes, coprocessor)

Factor of ~1.6 performance improvement by tuning.

Scaling measurements on BG/W (Blue Gene at Watson)

Weak scaling, relative performance per processor:

#nodes	coprocessor	virtual-node
512	1.000	0.974
1024	1.002	0.961
2048	0.985	0.963
4096	1.002	0.956
8192	1.009	0.935
16384	0.968	NAN

Outstanding scaling and exceptionally good use of the second cpu.

See an increase of time spent in `mpi_sendrecv` and `mpi_allreduce` at the largest process counts.

Remaining issue : results are NaNs for 32K cpus